Computer Cluster With Second-Node Instance of Application Having Access to State Snapshot of First-Node Instance of Application

[01] BACKGROUND OF THE INVENTION

[02] The present invention relates to computers and, more particularly, to a high-availability computer clusters (i.e., networks of computers that collaborate to minimize interruptions due to component failures). A major objective of the invention is to reduce the downtime associated with migrating a RAM-intensive application from a failed computer to an adoptive computer in a high-availability cluster.

[03] Modern society has been revolutionized by the increasing prevalence of computers. As computers have occupied increasingly central roles, their continued operation has become increasingly critical. For example, the cost of lengthy downtime for an on-line retailer during a peak shopping period can be unacceptable.

[04] Fault-tolerant computer systems have been developed that can continue running without any loss of data despite certain failure scenarios. However, fault-tolerant systems can be quite expensive. High-availability computer clusters have been developed as a more affordable alternative to fault-tolerant system to minimize downtime for mission-critical applications. For example, in a two-computer cluster with an independent hard disk system, data generated by an application running on a first computer can be stored on the hard disk system so that it is accessible by both computers. If the first computer fails, a dormant copy of the

application pre-installed on the second computer can be launched and have access to data on the hard disk system.

[05] Unlike fault-tolerant systems, high-availability systems can suffer data loss, e.g., data stored in volatile random-access memory or in processor registers and not otherwise saved to disk is typically lost when the host computer fails. However, if the initial data is stored on the hard disk system, the second instance of the application can recalculate the lost data. For many applications the additional delay involved in recalculating lost data can be quite small and acceptable.

[06] However, there are applications, such as supply-control-software (SCS), that run complex calculations in RAM for days. If a computer fails near the end of a calculation, days will be lost as a second instance of the program recalculates from initial conditions. Such programs often have a “save” function, so that the state of the program can be saved. However, due to the large amount of memory involved, e.g., eight gigabytes, the save can delay calculations considerably, e.g., an hour for each save operation. This extent of delay makes users reluctant to use the save function—and thus exposes them to major delays in the event of a computer failure. What is needed is a cluster system that reduces the amount of recalculation required upon a computer failure, without unduly delaying execution of the application.

[07] SUMMARY OF THE INVENTION

[08] The present invention provides for creating, in volatile (e.g., RAM) memory, a snapshot of the state of the first instance of an application running on a first computer of a computer. As the first instance is manipulating application data, the snapshot can be

transferred to storage media (e.g., a hard-disk system) accessible by a second instance of the application installed on a second computer in the same cluster. By initializing to the state represented by the snapshot, the second instance of the application can avoid having to repeat computations that led to the snapshot state. Preferably, the first computer is a multiprocessor system so that a processor not used for executing the application can be used for the transfer.

[09] In a conceptually simple realization of the invention, the application is "frozen" in a fixed state while the snapshot is created. In this case, a copy of the data in RAM is made and also stored in RAM. In addition, processor state information can be stored in RAM as part of the snapshot. Once the snapshot is created, the application can "unfreeze" and continue processing from the saved state. Since the state is saved to RAM instead of hard disk, the time the application must be frozen is much less than it would be if the state were saved directly to disk.

[10] Memory mapping techniques can avoid the need to create a complete copy of a state in RAM, further minimizing the impact of the snapshot on application execution. Initially, the snapshot can be represented by pointers to the memory being used by the application. Only memory sections about to be modified by the application need be copied; thus, the snapshot can "overlap" application memory to the extent that it remains unmodified. In the meantime, unmodified sections can be saved to disk. If a section to be modified has already been saved to disk, it need not be copied. Thus, RAM copying can be limited to sections of memory used by the application that are modified before they have been saved to disk. This, in turn, can be minimized by saving to disk first

unmodified sections of application memory, and then saving copied sections afterwards.

[11] In the event a failure halts execution of the application on the first computer, a second instance of the application on a second computer can access the snapshot and begin processing from the state represented in the snapshot rather than from initial conditions. If sufficient resources are available, the second instance of the application program can begin from the snapshot while the first instance is running; that way, if the first instance fails, the second instance will have progressed beyond the snapshot state, further reducing the delay suffered in the event of a failure.

[12] A major advantage of the invention is that upon a failure of a first instance of an application, a second instance can benefit from the processing accomplished by the first instance; the second instance does not have to start from initial conditions. Rather, the second instance can start from the state represented in a saved snapshot, or from a more advanced state. This advantage is achieved without requiring the application be halted as its state is saved to disk; in fact, the application need only be halted at times it seeks to modify sections of its memory not already saved to disk by a concurrent transfer-to-disk process. Furthermore, when it is so halted, it is only for as long as it takes to make a RAM copy of the section to be modified. Thus, the total delay suffered by the application can be much less than that required to create a copy of the memory used by the application in another area of RAM. These and other features and advantages of the invention are apparent from the description below with reference to the following figures.

[13] BRIEF DESCRIPTION OF THE DRAWINGS

[14] FIGURE 1 is a schematic diagram of a high-availability computer cluster in accordance with the present invention.

[15] FIGURE 2 is a flow chart of a method of the invention practiced in the context of the cluster of FIG. 1.

[16] DETAILED DESCRIPTION

[17] In accordance with the present invention, a computer cluster AP1 comprises a first computer CP1, a second computer CP2, a disk array MD1, and a network NW1. Computer CP1 includes a data processor DP1, a transfer processor DT1 (which can be identical to data processor DP1 but assigned to a different task herein), volatile random-access memory RM1, a root hard-disk RD1, a disk interface DI1, and a network interface NI1. Likewise, computer CP2 includes a data processor DP2, a transfer processor DT2, random-access memory RM2, a root hard-disk RD2, a disk interface DI2, and a network interface NI2.

[18] Disk interfaces DI1 and DI2 provide their respective host computers access to disk array MD1. Likewise, network interfaces NI1 and NI2 provide their respective host computers access to network NW1. Pursuant to the high-availability mission of computer cluster AP1, disk array RD1 is mirrored and network NW1 includes independent subnetworks.

[19] A first instance AA1 of a supply control application is installed and configured to run on computer CP1; a first instance AB1 of an accounting application is also installed on computer CP1 but is configured not to run. A second instance AB2 of the accounting application is installed and configured to run on

computer CP2. Also, a second instance AA2 of the supply control application is installed but not configured to run on computer CP2. In addition, cluster management daemons CD1 and CD2 are running respectively on computers CP1 and CP2.

[20] Since supply control application AA1 can process data for days in RAM RM1 to achieve a desired solution, a failure impacting computer CP1 can delay processing for a like time. Accordingly, cluster manager CM1 issues a "fork" command to processor DP1 while it is executing supply control application AA1. In response, processor DP1 creates a snapshot of the current process state of application AA1. Initially, this involves copying the state of processor DP1 to RAM RM1 and creating pointers to the sections of RAM RM1 being used in executing application AA1. Transfer processor DT1 is then used to transfer the contents of the addresses pointed to disk system MD1. Transferred sections are flagged as "transferred".

[21] While the snapshot is being transferred, processor DP1 continues execution of application AA1. When execution of application AA1 calls for modifying a section of memory included in the snapshot but that has not yet been transferred (as indicated by the transfer flag), the contents of the section are first copied to an unused section of RAM RM1, and the corresponding snapshot pointer is changed to point to the new location of the snapshot section. The original section can then be modified as required by application AA1 without affecting the integrity of the snapshot. If the application calls for modifying a section of memory that has already been transferred to disk, the section is not copied in RAM before being modifying.

[22] The resulting memory structure of RAM RM1 is shown in the detail of FIG. 1. RAM RM1 includes application data AD and snapshot data SD. At the instant the snapshot is first created, these two sets of data are coextensive except that the state data for processor DP1 is not part of application data AD. However, to the extent application AA1 calls for modifying sections of memory holding untransferred snapshot data, the two sets of data AD and SD diverge. The divergence defines distinct regions for unmodified application data ADU including uncopied snapshot data, modified application data ADM, copied memory snapshot data SNC, and processor snapshot data SNP. Snapshot data that has been transferred can be overwritten or discarded in RAM RM1.

[23] A method M1 of the invention is flow-charted in FIG. 1. The broken vertical line separates pre-failure and post-failure steps. At pre-failure step S1, supply control application AA1 is launched and executed on computer CP1. While application AA1 is running, cluster daemon CD1 causes a snapshot to be created of a processor and memory state associated with the execution of application AA1. This can involve copying processor registers and the application-related contents of the RAM to an unused region of RAM RM1. Preferably, however, the copying of RAM RM1 contents is performed only when a section is to be modified, as described above. To minimize copying in RAM RM1, uncopied snapshot data (corresponding to unmodified sections of application data) can be given transfer priority over copied snapshot data (corresponding to modified sections of application data).

[24] Once the snapshot is created, it is transferred to a shared disk system (or other nonvolatile memory accessible by computer CP2) at step S3A. Once transfer to the hard disk is completed, method M1

can return to step S2 and create a new snapshot, which is then transferred to disk. Preferably, the earlier snapshot would remain intact at least until writing of the later snapshot is complete; for example, successive snapshots can be written in alternation to two different file locations. Thus, a recent snapshot is always available on disk system MD1. Optionally, at step S3B, computer CP2 executes the second instance AA2 of the supply control application from the state represented in the snapshot even in the absence of a failure. Method M1 then has two variants M1A and M1B, depending on whether step S3B is implemented.

[25] In variant M1A, upon detection at step S4A of a failure that prevents execution of the first instance AA1 of the supply-control application on computer CP1, the second instance AA2 of the application can be initialized using the most-recent snapshot state and then permitted to execute from that state at step S5A. In variant M1B, upon a comparable failure detection at step S4B, there is no need to initialize second instance AA2 as it is already executing and has progressed beyond the most-recent snapshot. Instead, at step S5B, second instance AA2 is treated as the main instance. For example, after the failure detection of step S4B, snapshots can be taken of the state of instance AA2, whereas before step S4B, this would not have been done.

[26] Variant M1B provides for further reducing the maximum delay by beginning execution from a snapshot state as soon as it is available on hard disk system MD1 without the need for an intervening failure. Thus, if a failure occurs, the second instance AA2 will have advanced beyond the snapshot state and the lost time will be reduced correspondingly. However, as computer CP2 is also running accounting application AB2, the resources available to

application AA2 are less than they were pre-failure for instance AA1. Hence, processing of instance AA2 can be slower than for instance AA1. Accordingly, there is still some advantage to providing updated snapshots from instance AA1. On the other hand, variant M1A can be preferred because it is less demanding on resources and can permit higher performance execution of application AB2.

[27] In the preferred embodiment, the snapshot is transferred from RAM to a shared hard-disk system; alternatively, it can be transferred from RAM to a local hard disk of the first computer (or to shared disk storage) and then to a local disk of the adopting computer. Alternatively, the transfer can be from RAM directly to a local disk of the adopting computer.

[28] Snapshot data in RAM RM1 can be discarded once it has been transferred to disk. This can involve discarding pointers to copied and uncopied snapshot data as the snapshot data is transferred or once transfer is complete. An intermediate solution is to retain snapshot data until transfer is complete except that transferred snapshot data is discarded to the extent it is overwritten by application data.

[29] While the foregoing description involves a cluster running a supply-control application and an accounting application, the invention applies to a wide variety of applications. The invention provides the greatest advantage for RAM-intensive applications such as logistics, artificial intelligence (e.g., as applied to data analysis), and various scientific applications. While the invention provides the greatest advantage to RAM-intensive applications, it applies more generally to applications that do a lot of processing between saves

to disk. Furthermore, the invention can provide a performance alternative to many applications that do save to disk often.

[30] The invention provides a variety of approaches to handling successive snapshots. For example, each snapshot could overwrite its predecessor; however, this might leave the system without an intact snapshot in the event a failure occurs during overwriting. To ensure continuous availability of an intact snapshot, a succeeding snapshot is preferably written to a different location than its predecessor. The snapshots can be given different file names, or a succeeding snapshot can be written to a temporary file that assumes a single snapshot file name once writing is complete. Alternatively, a backup copy of an earlier snapshot can remain intact while a main copy is overwritten. Those skilled in the art are aware of many alternatives to ensuring an intact file is continuously maintained while more recent versions are stored. In this vein, the storage media used for storing snapshots can be a single hard disk, a multi-disk system, and/or a distributed set of disks or other storage media.

[31] The invention provides alternative ways of configuring a two-computer cluster. For example, an application can run from an initial state on both computers, while snapshots are taken from one computer and stored in case of failure, while the other computer provides maximum performance by not creating snapshots. Alternatively, two instances of an application can run from an initial state, with snapshots being taken in alternation from the two computers.

[32] The invention also applies to clusters of three and more computers, including clusters of clusters such as those used for disaster-tolerant applications. For example, an application can run

on two nodes, each of which provides snapshots to a third backup node. Alternatively, one node can provide snapshots to in alternation to two disk arrays for access by different potential adoptive nodes. These and other variations upon and modifications to the detailed embodiment are provided for by the present invention, the scope of which is defined by the following claims.

[33] **What Is Claimed Is:**